# Discrete Mathematics in Computer Science

## Fibonacci Series – Generating Functions

Malte Helmert, Gabriele Röger

University of Basel

# Revisiting the Fibonacci Series

- In this section we study generating functions, a powerful method for solving recurrences.
- Generating functions allow us to directly derive closed-form expressions for recurrences.
- Full mastery of generating functions requires solid knowledge of calculus, in particular power series.
- Rather than develop the topic in its full depth, we will look at it within the context of a case study, proving the closed form of the Fibonacci series again.
- We leave out some of the more subtle mathematical aspects, such as the question of convergence of the power series used.

# Power Series

**Definition (power series)**

Let $(a_n)_{n \in \mathbb{N}_0}$ be a sequence of real numbers.
The power series with coefficients $(a_n)$ is the (possibly partial) function $g : \mathbb{R} \to \mathbb{R}$ defined by

$$g(x) = \sum_{n=0}^{\infty} a_n x^n \qquad \text{for all } x \in \mathbb{R}.$$

German: Potenzreihe

Notes: more general definitions exist, for example

- using $(x - c)^n$ instead of $x^n$ for some $c \in \mathbb{R}$
- using complex instead of real numbers
- using multiple variables

# Power Series – Examples

Reminder: $g(x) = \sum_{n=0}^{\infty} a_n x^n$

Examples:

- $a_n = 1$
  $\rightsquigarrow g(x) = \frac{1}{1-x}$ (only defined for $|x| < 1$)
- $a_n = z^n$ for some $z \in \mathbb{R}$
  $\rightsquigarrow g(x) = \frac{1}{1-zx}$ (only defined for $|x| < 1/|z|$)
- $a_n = \frac{1}{n!}$
  $\rightsquigarrow g(x) = e^x$ (defined for all $x$)
- $a_n = \begin{cases} 0 & x \text{ is even} \\ \frac{(-1)^{(n-1)/2}}{n!} & x \text{ is odd} \end{cases}$
  $\rightsquigarrow g(x) = \sin x$ (defined for all $x$)

# Uniqueness of Power Series Representation

**Theorem**

*Let $g$ and $h$ be power series with coefficients $(a_n)$ and $(b_n)$.
Let $\varepsilon > 0$ such that for all $|x| < \varepsilon$:*

- *$g$ and $h$ converge, and*
- *$g(x) = h(x)$.*

*Then $a_n = b_n$ for all $n \in \mathbb{N}_0$.*

# Generating Functions

**Definition (generating function)**

Let $f : \mathbb{N}_0 \to \mathbb{R}$ be a function over the natural numbers.
The generating function for $f$ is the power series
with coefficients $(f(n))_{n \in \mathbb{N}_0}$.

German: erzeugende Funktion

We are particularly interested in the case where $f$ is defined by a recurrence.

# Generating Functions for Solving Recurrences

General approach for deriving closed-form expressions for a recurrence $f$ using generating functions:

1. Let $g$ be the generating function of $f$.
2. Use the recurrence to derive an equation for $g$.
3. Use algebra and calculus to solve the equation, i.e., derive a closed-form expression for $g$.
4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.
5. We get $f(n) = a_n$ as the closed-form expression of the recurrence.

# Case Study: Fibonacci Numbers

We now illustrate the approach using the Fibonacci numbers $F$ as an example for the recurrence $f$.

As a reminder, the Fibonacci numbers are defined as follows:

- $F(0) = 0$
- $F(1) = 1$
- $F(n) = F(n-1) + F(n-2)$ for all $n \geq 2$

# Case Study: 1. Generating Function

1. Let $g$ be the generating function of $f$.

$$g(x) = \sum_{n=0}^{\infty} F(n)x^n \quad \text{for } x \in \mathbb{R}$$

Note: The series does not converge for all $x$, but it converges for $|x| < \varepsilon$ for sufficiently small $\varepsilon > 0$. We omit details.

# Case Study: 2. Equation for $g$ from Recurrence

$F(0) = 0 \qquad F(1) = 1 \qquad F(n) = F(n-1) + F(n-2)$ for all $n \geq 2$

2. Use the recurrence to derive an equation for $g$.

$$
\begin{aligned}
g(x) = \sum_{n=0}^{\infty} F(n)x^n &= 0 \cdot x^0 + 1 \cdot x^1 + \sum_{n=2}^{\infty}(F(n-1) + F(n-2))x^n \\
&= x + \sum_{n=2}^{\infty} F(n-1)x^n + \sum_{n=2}^{\infty} F(n-2)x^n \\
&= x + \sum_{n=1}^{\infty} F(n)x^{n+1} + \sum_{n=0}^{\infty} F(n)x^{n+2} \\
&= x + x\sum_{n=1}^{\infty} F(n)x^n + x^2\sum_{n=0}^{\infty} F(n)x^n \\
&= x + x\sum_{n=0}^{\infty} F(n)x^n + x^2\sum_{n=0}^{\infty} F(n)x^n \\
&= x + x\,g(x) + x^2 g(x)
\end{aligned}
$$

# Case Study: 3. Solve Equation for $g$

3. Use algebra and calculus to solve the equation, i.e., derive a closed-form expression for $g$.

$$
\begin{aligned}
g(x) &= x + x\,g(x) + x^2 g(x) \\
\Rightarrow \quad g(x) - x\,g(x) - x^2 g(x) &= x \\
\Rightarrow \quad g(x)(1 - x - x^2) &= x \\
\Rightarrow \quad g(x) &= \frac{x}{1 - x - x^2}
\end{aligned}
$$

# Case Study: 4. Power Series Representation for $g$ (1)

4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.

$g(x) = \frac{x}{1-x-x^2} = xh(x)$ with $h(x) = \frac{1}{1-x-x^2}$

Idea: partial fraction decomposition, i.e.,
find $a, b, \alpha, \beta$ such that $h(x) = \frac{a}{1-\alpha x} + \frac{b}{1-\beta x}$.

$$\begin{aligned}\frac{a}{1-\alpha x} + \frac{b}{1-\beta x} &= \frac{a(1-\beta x) + b(1-\alpha x)}{(1-\alpha x)(1-\beta x)} \\ &= \frac{a - a\beta x + b - b\alpha x}{1 - \alpha x - \beta x + \alpha\beta x^2} \\ &= \frac{(a+b) + (-a\beta - b\alpha)x}{1 + (-\alpha - \beta)x + \alpha\beta x^2}\end{aligned}$$

$\rightsquigarrow a + b = 1, \quad -a\beta - b\alpha = 0, \quad -\alpha - \beta = -1, \quad \alpha\beta = -1$

# Case Study: 4. Power Series Representation for $g$ (2)

4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.

(1) $a + b = 1$, (2) $-a\beta - b\alpha = 0$, (3) $-\alpha - \beta = -1$, (4) $\alpha\beta = -1$

- From (3): (5) $\beta = 1 - \alpha$
- Substituting (5) into (4):

$$
\begin{aligned}
& \alpha(1-\alpha) = -1 \\
\Rightarrow\quad & \alpha - \alpha^2 = -1 \\
\Rightarrow\quad & \alpha^2 - \alpha - 1 = 0 \\
\Rightarrow\quad & \alpha = \frac{1}{2} \pm \sqrt{\frac{1}{4} + 1} = \frac{1}{2} \pm \sqrt{\frac{5}{4}} \\
\Rightarrow\quad & \alpha = \frac{1 \pm \sqrt{5}}{2}
\end{aligned}
$$

$\rightsquigarrow$ The solutions are $\alpha = \varphi$ or $\alpha = \psi$ from the previous chapter. Continue with (6) $\alpha = \varphi$.

# Case Study: 4. Power Series Representation for $g$ (3)

4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.

(1) $a + b = 1$, (2) $-a\beta - b\alpha = 0$, (3) $-\alpha - \beta = -1$, (4) $\alpha\beta = -1$,
(5) $\beta = 1 - \alpha$, (6) $\alpha = \varphi$

- Substituting (6) into (5): (7) $\beta = 1 - \alpha = 1 - \varphi = \psi$.
- From (1): (8) $b = 1 - a$
- Substituting (6), (7), (8) into (2):

$$
\begin{aligned}
& -a(1-\varphi) - (1-a)\varphi = 0 \\
\Rightarrow \quad & -a + a\varphi - \varphi + a\varphi = 0 \\
\Rightarrow \quad & a(2\varphi - 1) = \varphi \\
\Rightarrow \quad & a = \frac{\varphi}{2\varphi - 1} = \frac{\varphi}{2 \cdot \frac{1}{2}(1+\sqrt{5}) - 1} = \frac{1}{\sqrt{5}}\varphi
\end{aligned}
$$

# Case Study: 4. Power Series Representation for $g$ (4)

4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.

(8) $b = 1 - a$, (9) $a = \frac{1}{\sqrt{5}}\varphi$

- Substituting (9) into (8):

$$
\begin{aligned}
b &= 1 - a \\
&= 1 - \frac{1}{\sqrt{5}}\varphi \\
&= \frac{\sqrt{5}}{\sqrt{5}} - \frac{\frac{1}{2}(1+\sqrt{5})}{\sqrt{5}} \\
&= -\frac{1}{\sqrt{5}}(-\sqrt{5} + \frac{1}{2} + \frac{1}{2}\sqrt{5}) \\
&= -\frac{1}{\sqrt{5}}(\frac{1}{2} - \frac{1}{2}\sqrt{5}) \\
&= -\frac{1}{\sqrt{5}}\psi
\end{aligned}
$$

# Case Study: 4. Power Series Representation for $g$ (5)

4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.

$g(x) = xh(x), \quad h(x) = \frac{a}{1-\alpha x} + \frac{b}{1-\beta x},$

$\alpha = \varphi, \quad \beta = \psi, \quad a = \frac{1}{\sqrt{5}}\varphi, \quad b = -\frac{1}{\sqrt{5}}\psi$

Plugging everything in:

$$
\begin{aligned}
g(x) &= x\left(\frac{1}{\sqrt{5}}\varphi\frac{1}{1-\varphi x} - \frac{1}{\sqrt{5}}\psi\frac{1}{1-\psi x}\right) = \frac{x}{\sqrt{5}}\left(\varphi\frac{1}{1-\varphi x} - \psi\frac{1}{1-\psi x}\right) \\
&= \frac{x}{\sqrt{5}}\left(\varphi\sum_{n=0}^{\infty}\varphi^n x^n - \psi\sum_{n=0}^{\infty}\psi^n x^n\right) \\
&= \frac{1}{\sqrt{5}}\left(\sum_{n=0}^{\infty}\varphi^{n+1}x^{n+1} - \sum_{n=0}^{\infty}\psi^{n+1}x^{n+1}\right) \\
&= \frac{1}{\sqrt{5}}\left(\sum_{n=1}^{\infty}\varphi^n x^n - \sum_{n=1}^{\infty}\psi^n x^n\right) = \sum_{n=1}^{\infty}\frac{1}{\sqrt{5}}(\varphi^n - \psi^n)x^n \\
&= \sum_{n=0}^{\infty}\frac{1}{\sqrt{5}}(\varphi^n - \psi^n)x^n
\end{aligned}
$$

# Case Study: 5. Extract Closed Form of Recurrence

4. Use calculus to derive a power series representation $\sum_{n=0}^{\infty} a_n x^n$ for $g$.
5. We get $f(n) = a_n$ as the closed-form expression of the recurrence.

From

$$g(x) = \sum_{n=0}^{\infty} \frac{1}{\sqrt{5}}(\varphi^n - \psi^n)x^n$$

we conclude:

$$F(n) = \frac{1}{\sqrt{5}}(\varphi^n - \psi^n) \quad \text{for all } n \in \mathbb{N}_0$$

# Concluding Remarks

- The approach requires analytical skill, but once understood, it can be applied to many similar recurrences.
- The same basic idea can be used to solve all recurrences of the form
  - $f(0) = a_0$
  - ...
  - $f(k-1) = a_{k-1}$
  - $f(n) = c_1 f(n-1) + \cdots + c_k f(n-k) \quad$ for all $n \geq k$
- The Fibonacci numbers are the special case where $k = 2$, $a_0 = 0$, $a_1 = 1$, $c_1 = 1$, $c_2 = 1$.

# Discrete Mathematics in Computer Science

## Master Theorem for Divide-and-Conquer Recurrences

Malte Helmert, Gabriele Röger

University of Basel

# Divide-and-Conquer Algorithms

- Recurrences frequently arise in the run-time analysis of divide-and-conquer algorithms.
- Examples:
  - Mergesort: sort a sequence by recursively sorting two smaller sequences, then merging them
  - Binary search: find an element in a sorted sequence by identifying which half of the sequence must contain the element, then recursively searching it
  - Quickselect: find the $k$-th smallest element in a sequence by recursive partitioning

# Asymptotic Growth

- Run-time analysis usually focuses on the asymptotic growth rate of run-time.
- For example, we say "run-time grows at most quadratically" rather than saying that run-time for inputs of size $n$ is $3n^2 + 17n + 8$.

advantages:

- much simpler to study
- can abstract from minor implementation details

# Big-$O$, Big-$\Omega$, Big-$\Theta$

**Definition ($O$, $\Omega$, $\Theta$)**

Let $g : \mathbb{R}_0^+ \to \mathbb{R}$ be a function.
The sets of functions $O(g), \Omega(g), \Theta(g)$ are defined as follows:

- $O(g) = \{f : \mathbb{R}_0^+ \to \mathbb{R} \mid$ there exist $C, n_0 \in \mathbb{R}$ s.t. $|f(n)| \leq C \cdot g(n)$ for all $n \geq n_0\}$
- $\Omega(g) = \{f : \mathbb{R}_0^+ \to \mathbb{R} \mid$ there exist $C, n_0 \in \mathbb{R}$ s.t. $|f(n)| \geq C \cdot g(n)$ for all $n \geq n_0\}$
- $\Theta(g) = O(g) \cap \Omega(g)$

Notation:

- It is convention to say "$5n^2 + 7n \log_2 n = \Theta(n^2)$" instead of "$f \in \Theta(g)$ for the functions $f$, $g$ with $f(n) = 5n^2 + 7n \log_2 n$ and $g(n) = n^2$".
- ditto for $O$, $\Omega$

# Divide-and-Conquer Recurrences

A common instantiation of the divide-and-conquer algorithm scheme works as follows:

- For inputs of small size $n < C$, solve the problem directly.
- Otherwise:
  1. Construct $A$ smaller inputs of size $n/B$.
  2. Recursively solve these inputs using the same algorithm.
  3. Compute the result from the recursively computed results.

If 1.+3. take time $f(n)$, the overall run-time for $n > C$ can be expressed as $T(n) = A \cdot T(n/B) + f(n)$.

- We call this a divide-and-conquer recurrence.
- We do not care about run-time for $n \leq C$ because it does not affect asymptotic analysis.

# Divide-and-Conquer Recurrences – Examples

Reminder:

1. Construct $A$ smaller inputs of size $n/B$.
2. Recursively solve these inputs using the same algorithm.
3. Compute the result from the recursively computed results.

divide-and-conquer recurrence: $T(n) = A \cdot T(n/B) + f(n)$

Examples:

- Mergesort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
- Binary Search: $A = 1,\ B = 2,\ f(n) = \Theta(1)$

# Master Theorem for Divide-and-Conquer Recurrences

**Theorem**

*Let $A \geq 1, B \geq 1$, and let $T$ satisfy the divide-and-conquer recurrence $T(n) = A \cdot T(n/B) + f(n)$. Then:*

- *If $f(n) = O(n^{\log_B A - \varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_B A})$.*
- *If $f(n) = \Theta(n^{\log_B A})$, then $T(n) = \Theta(n^{\log_B A} \log_2 n)$.*
- *If $f(n) = \Omega(n^{\log_B A + \varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(f(n))$.*

We do not prove the theorem.

# Application: Mergesort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Mergesort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$

# Application: Mergesort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Mergesort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$

$\rightsquigarrow \log_B A = \log_2 2 = 1$

# Application: Mergesort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Mergesort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- $f(n) = O(n^{1-\varepsilon}) \rightsquigarrow T(n) = \Theta(n^1)$
- $f(n) = \Theta(n^1) \rightsquigarrow T(n) = \Theta(n^1 \log_2 n)$
- $f(n) = \Omega(n^{1+\varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

# Application: Mergesort

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Mergesort: $A = 2,\ B = 2,\ f(n) = \Theta(n)$
$\rightsquigarrow \log_B A = \log_2 2 = 1$

- $f(n) = O(n^{1-\varepsilon}) \rightsquigarrow T(n) = \Theta(n^1)$
- $f(n) = \Theta(n^1) \rightsquigarrow T(n) = \Theta(n^1 \log_2 n)$
- $f(n) = \Omega(n^{1+\varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

$\rightsquigarrow T(n) = \Theta(n \log n)$

# Application: Binary Search

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Binary Search: $A = 1,\ B = 2,\ f(n) = \Theta(1)$

# Application: Binary Search

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Binary Search: $A = 1,\ B = 2,\ f(n) = \Theta(1)$
$\rightsquigarrow \log_B A = \log_2 1 = 0$

# Application: Binary Search

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Binary Search: $A = 1,\ B = 2,\ f(n) = \Theta(1)$
$\rightsquigarrow \log_B A = \log_2 1 = 0$

- $f(n) = O(n^{0-\varepsilon}) \rightsquigarrow T(n) = \Theta(n^0)$
- $f(n) = \Theta(n^0) \rightsquigarrow T(n) = \Theta(n^0 \log_2 n)$
- $f(n) = \Omega(n^{0+\varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

# Application: Binary Search

Reminder: $T(n) = A \cdot T(n/B) + f(n)$

- $f(n) = O(n^{\log_B A - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^{\log_B A})$
- $f(n) = \Theta(n^{\log_B A}) \rightsquigarrow T(n) = \Theta(n^{\log_B A} \log_2 n)$
- $f(n) = \Omega(n^{\log_B A + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

Binary Search: $A = 1,\ B = 2,\ f(n) = \Theta(1)$
$\rightsquigarrow \log_B A = \log_2 1 = 0$

- $f(n) = O(n^{0 - \varepsilon}) \rightsquigarrow T(n) = \Theta(n^0)$
- $f(n) = \Theta(n^0) \rightsquigarrow T(n) = \Theta(n^0 \log_2 n)$
- $f(n) = \Omega(n^{0 + \varepsilon}) \rightsquigarrow T(n) = \Theta(f(n))$

$\rightsquigarrow T(n) = \Theta(\log n)$

# More Complex Cases

Some divide-and-conquer algorithms have more complicated recurrences because they do not split into even-sized pieces of predictable size.

Example:

- Quicksort with random pivotization: $f(n) = \Theta(n)$;
  split $n$ uniformly randomly into $1 \leq k \leq n$ and $n - 1 - k$
  $\rightsquigarrow$ expected runtime $\Theta(n \log n)$
- Quickselect with median-of-median pivotization: $f(n) = \Theta(n)$;
  one recursion on input size $n/5$,
  one recursion on input size at most $n \cdot \frac{7}{10}$
  $\rightsquigarrow$ runtime $\Theta(n)$

Here, we can try to use the Master theorem to derive hypotheses and then prove them by mathematical induction.